1. 概述

dos命令行:

dir：列出所有文件以及文件夹

md:创建目录

rd：删除目录

cd：进入指定目录

cd..：回到上一个目录

cd/：回到根目录

del：删除文件

exit:退出cmd

环境的配置：

1. 我的电脑--属性--高级--环境变量
2. 编辑path环境变量，在开始处加上java所在目录
3. 在dos命令行中敲入javac，如果跳出参数信息则说明成功

临时环境变量：

1. Dos中的set命令
   1. set：用于查看本机中的所有环境变量
   2. set 变量名： 用于查看具体一个环境变量的值
   3. set 变量名 = ：清空一个环境变量
   4. set 变量名 = 具体值 ： 给具体变量定义具体值
2. 想要在原有环境变量基础上添加新值
   1. 首先，通过 %变量名% 获取原来的
   2. 然后在给变量植入新值

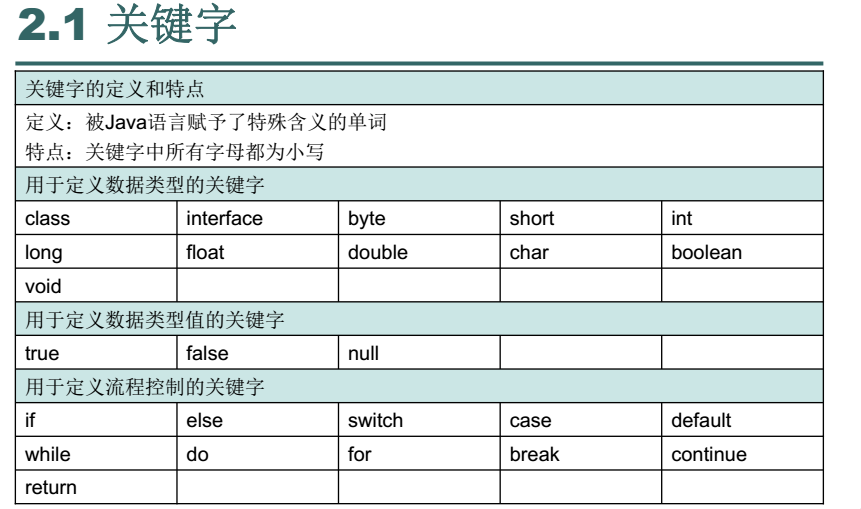
例子： set path = 新值；%path%

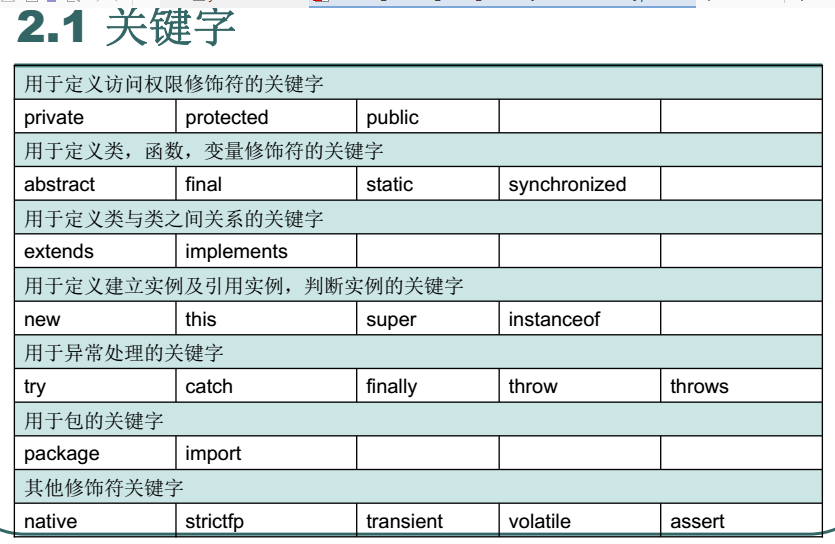
注意：关闭dos窗口就失效

1. 为了不因为jdk的目录或者目录名称的改变，而不断改变path值，而导致对path变量的误操作，可以用以下方法完成
   1. 新建一个环境变量 Java\_home 记住jdk目录
   2. 在path中通过%%动态的获取java\_home的值
   3. Java\_home=F:\jdk.1.321
   4. Path=%java\_home%\bin;%path%

%path% 动态获取path环境变量的值

%java\_home% 动态获取名称为java\_home环境变量的值





函数定义的格式：

修饰符 返回值类型 函数名(参数类型 形式参数1，参数类型 形式参数2...)

{

执行语句；

return 返回值；

数组的定义格式：

//这两个是一致的 第一种定义方法

// int[] arr = new int[2]; //推荐用这个

// int arr[] = new int[2];

//第二种定义方法

// int[] arr = new int[]{3,1,6,5,4}; //静态初始化方式 []这里别写长度 {}写内容

// int[] arr = {3,1,6,5,4};//简化形式

// System.out.println(arr[2]); // =6

//不确定值的情况

int[] arr = new int[3];

// System.out.println(arr[3]);

//ArrayIndexOutOfBoundsException:3: 操作数组时，访问到了数组中不存在的角标 越界了

arr = null;

System.out.println(arr[1]);

//NullPointerException:空指针异常：当引用没有任何指向值位null的情况，该引用还在操作 //用于操作实体

//冒泡排序 面试常考

public static void bubbleSort(int[] arr)

{

for(int x=0; x<arr.length-1; x++)

{

for(int y=0; y<arr.length-x-1; y++) // -x:让每一次比较的元素减少，

//-1：避免角标越界

{

if(arr[y]>arr[y+1])

{

/\*

int temp = arr[y];

arr[y] = arr[y+1];

arr[y+1] = temp;

\*/

swap(arr,x,y);

}

}

}

}

构造代码块：

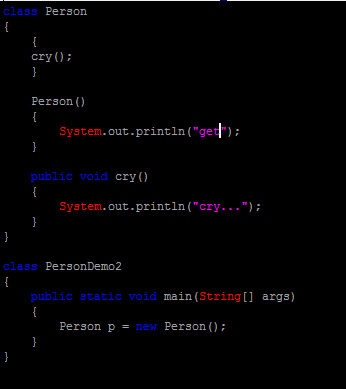
作用：给对象进行初始化

对象一建立就运行，而且优先于构造函数执行。

和构造函数的区别：

构造代码块是给所有对象进行统一初始化

而构造函数是给对应的对象初始化



这样：其中

{

Cry();

}

Person()

{

}

都是直接被运行的，不用调 用

输出结果是get

cry...

this的用法:

this：看上去是用于区分局部变量和成员变量同名的情况

this为什么可以解决这个问题？

this到底代表的是什么呢？

this：就代表本类的对象，到底代表哪一个呢？

this代表它所在函数所属对象的引用

简单说：哪个对象在调用this所在的函数，this就代表哪个对象

this的应用：当定义类中功能时，该函数内部要用到调用该函数的对象时，这时用this来表示这个对象

但凡本类功能内部使用了了本类对象，都用this表示。

this语句：用于构造函数之间进行互相调用

this语句只能定义在构造函数的第一行。因为初始化要先执行。

主函数:

public static void main(string[] args)

主函数：是一个特殊的函数，作为程序的入口，可以被jvm调用

主函数的定义：

public：代表着该函数的访问权限是最大的

static：代表着主函数随着类的加载就已经存在了；

void:主函数没有具体的返回值

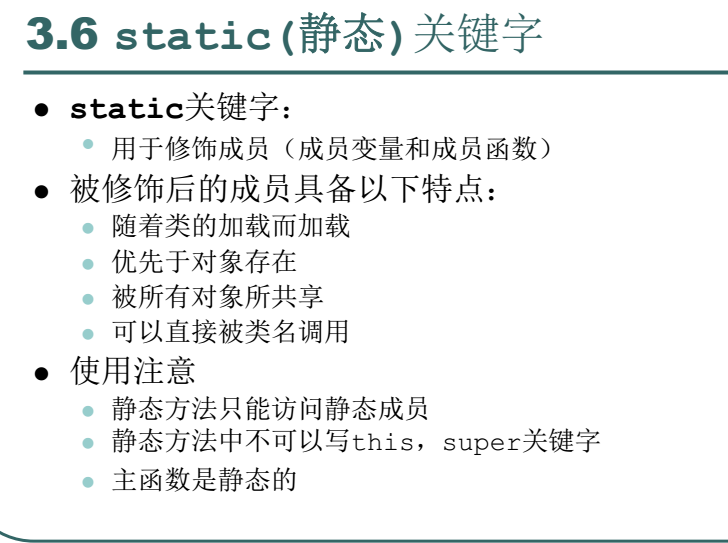
main：不是关键字，但是是一个特殊的单词，可以被jvm识别

函数的参数:（String[] args）

(String[] arr):函数的参数，参数类型是一个数组，该数组中的元素是字符串，字符串类型的数组

主函数是固定格式的，jvm识别。

jvm在调用主函数时，传入的是new String[0]



静态代码块.

格式：

static

{

静态代码块中的执行语句。

}

特点：随着类的加载而执行，只执行一次,并优先于主函数

用于给类进行初始化的。

静态：static

用法：是一个修饰符，用于修饰成员（成员变量，成员函数）

当成员被静态修饰后，就多了一个调用方式，除了可以被对象调用外，

还可以直接被类名调用。类名.静态成员

static特点：

1.随着类的加载而加载

也就是说：静态会随着类的消失而消失，说明它的生命周期最长

2.优先于的对象存在

明确一点：静态是先存在，对象是后存在的。

3.被所有对象所共享

4.可以直接被类名所调用。

实例变量和类变量的区别：

1.存放位置。

类变量随着类的加载而存在于方法区中。

实例变量随着对象的建立而存在于堆内存中

2.生命周期：

类变量生命周期最长，随着类的消失而消失

实例变量生命周期随着对象的消失而消失

3.主函数是静态的

静态的使用注意事项:

1.静态方法只能访问静态成员

非静态既可以访问静态也可以访问非静态

2.静态方法中不可以定义this.super关键字

因为静态优先于对象存在，所以静态中不可以出现this

静态有利有弊

利：对对象的共享数据进行单独空间的存储，节省空间。

可以直接被类名调用。

弊：生命周期过长。访问出现局限性

例子：

class Person

{

private String name;

private int age;

private static String country = "cn";

Person(String name,int age)

{

this.name = name;

this.age = age;

}

{

System.out.println(name+".."+age);

}

public void setName(String name)

{

this.name = name;

}

public void speak()

{

System.out.println(this.name+".."+this.age);

}

public static void showCountry()

{

System.out.println("country="country);

}

}

class PersonDemo

{

public static void main(String[] args)

{

Person p = new Person("zhangsan",20);

p.setName("Lisi");

}

}

/\*

Person p = new Person("zhangsan",20);

这句话做了什么事情

1.因为new用到了Person.class 所以会先找到Person.class文件并加载到内存中

2.执行该类中的static代码块，如果有的话，给Person.class类进行初始化

3.在堆内存中开辟空间，分配内存地址。

4.在堆内存中建立对象的特有属性。并进行默认初始化

5.对属性进行显示初始化

6.对对象进行改造代码块初始化。

7.对对象进行对应的构造函数初始化

8.讲内存地址赋给栈内存中p变量

什么时候使用静态

两个方面：

因为静态修饰的内容有成员变量和函数。

什么时候定义静态变量（类变量）呢？

答：当对象中出现共享数据时，该数据被静态所修饰

对象中的特有数据要定义成非静态存在于堆内存中（对象内部）。

什么时候定义静态函数呢？

答：当功能内部没有访问到非静态数据（对象的特有数据），

那么该功能可以定义成静态的

抽象 abstract

当多个类中出现相同功能，但功能主体不同

这是可以进行向上抽取，这时，只抽取功能定义，而不抽取功能主体

抽象就是看不懂

抽象类的特点：

1.抽象方法一定在抽象类中

2.抽象方法和抽象类都必须被abstract关键字修饰

3.抽象类不可以用new创建对象。因为调用抽象方法没意义。

4.抽象类中的方法要被使用，必须由子类复写起所有的抽象方法后，建立子类对象调用

如果子类只覆盖了部分抽象方法，那么该子类还是一个抽象类

抽象类和一般类没有太大的不同

该怎么描述事物就怎么描述事物，只不过，该事物出现了一些看不懂的东西。

这些不确定的部分，也是该事物的功能，需要明确出现，但是无法定义主体

通过抽象方法表示。

抽象类比一般类多了个抽象函数。 就是在类中可以定义抽象方法。

抽象类不可以实例化。

特殊：抽象类中可以不定义抽象方法，这样做仅仅是不让该类建立对象·

继承： is a （extends）

1.提高代码的复用性

2.让类与类之间产生了关系，有了这个关系，才有了多态的特性。

注意：千万不要为了获取其他类的功能，简化代码而继承。

必须是类与类之间有所属关系才可以继承

在java语言中：java只支持单继承，不支持多继承

因为多继承容易带来安全隐患：当多个父类中定义了相同功能，当功能内容不同时不确定要用哪一个

但是java保留这种机制，并用另一种体现形式来完成表示。叫作多实现

java支持多层继承，也就是一个继承体系

如何使用一个继承体系中的功能呢？

想要使用体系，先查阅体系父类的描述，因为父类中定义的是该体系中共性功能

通过了解共性功能，就可以知道该体系的基本功能。

那么这个体系已经可以基本使用了

那么在具体调用时，要创建最子类的对象，

1.因为有可能父类不能创建对象

2.创建子类对象可以使用更多的功能，包括基本的，也包括特有的

简单一句话：查阅父类功能，创建子类对象使用功能

除了继承还有其他关系：

聚集： has a

聚合：

组合：

/\*

子父类出现后，类成员的特点

类中成员：

1.变量

2.函数

3.构造函数

1.变量

如果子类中出现非私有的同名成员变量时。

子类要访问本类中的变量，用this

**子类要访问父类中的用super**

2.子父类中的函数

当子类出现和父类一模一样的函数时

当子类对象调用该函数，会运行子类函数的内容

如同父类的函数被覆盖一样

这种情况是函数的另一个特性:重写（覆盖）

当子类继承父类，沿袭了父类的功能到子类中

但是子类虽具备该功能，但是功能的内容却和父类不一致

这时，没有必要定义新功能，而是使用覆盖特性，保留父类的功能定义，并重写功能

覆盖：

1.子类覆盖父类，必须保证子类权限大于等于父类权限，才可以覆盖，否则编译失败

2.静态只能覆盖静态。

重载：只看同名函数的参数列表

重写：子父类方法要一模一样

3.子父类中的构造函数

在对子类对象进行初始化时，父类的构造函数也会运行

那是因为子类构造函数默认第一行有一天隐藏的语句 super()；

super()：会访问父类中空参数的构造函数，而且子类中所有构造函数默认第一行都是super()

为什么子类一定要访问父类中的构造函数.

因为父类中的数据子类可以直接获取，所以子类对象在建立时，需要先查看父类是如何对这些数据进行初始化的

所以子类在对象初始化时，要先访问一下父类中的构造函数。

如果要访问父类中指定的构造函数，可以通过手动定义super语句的方式来指定。

注意：super语句一定定义在子类构造函数的第一行

子类的实例化过程：

结论：

子类的所有的构造函数，默认都会访问父类中空参数的构造函数。

因为子类每一个构造函数内的第一行都有一句隐藏的super()

当父类中没有空参数的构造函数时，子类必须手动通过super或者this语句形式来指定要访问父类中的构造函数。

当然：子类的构造函数第一行也可以手动指定this语句来访问本类中的构造函数，子类中至少有一个构造函数会访问父类中的构造函数

final:最终， 作为一个修饰符

1.可以修饰类、函数、变量。

2.被final修饰的类不可以被继承。为了避免被继承，被子类复写功能

3.被final修饰的方法不可以被复写

4.被final修饰的变量是一个常量只能赋值一次，既可以修饰成员变量，又可以修饰局部变量

当在描述事物时，一些数据的出现值是固定的，那么这时为了增强阅读性，都给这些值起个名字，方便于阅读

而这个值不需要改变，就加final修饰.作为常量：常量的书写规范所有字母都大写，如果由多个单词组成，

单词间通过\_连接

内部类定义在类中的局部位置上时，只能访问该局部被final修饰的局部变量

接口：初期理解，可以认为是一个特殊的抽象类(interface)

当抽象类中的方法都是抽象的，那么该类可以通过接口的形式来表示

class 用于定义类

interface用于定义接口

接口定义时，格式特点：

1.接口中常见定义：常量，抽象方法

2.接口中的成员都有固定的修饰符

常量：public static final

方法：public abstract

接口中的成员都是public的

接口是不可以创建对象的，因为有抽象方法。

需要被子类实习，子类对象接口中的抽象方法全都覆盖后，子类才可以实例化。

否则子类是一个抽象类

接口可以被类多实现。也是对多继承不支持转换形式。java支持多实现

implements是实现的意思 类似于extends

多态：可以理解为事物存在的多种体现形态

人：man、woman

动物:猫、狗

猫 x = new 猫();

动物 x = new 猫();

1.多态的体现

父类的引用指向了自己的子类对象。

父类的引用也可以接受自己的子类对象。

2.多态的前提

必须是类与类之间有关系，要么继承，要么实现。

通常还有一个前提:存在覆盖

3.多态的好处

多态的出现大大的提高了程序的扩展性。

4.多态的弊端

提高了扩展性，但是只能使用父类的引用访问父类中的成员。

5.多态的应用

在多态中成员函数的特点:

在编译时期：参阅引用型变量所属的类中是否有调用的方法。如果有，编译通过，如果没有编译失败

在运行时期：参阅对象所属的类中是否有调用的方法。

简单总结就是：成员函数在多态调用时，编译看左边，运行看右边

在多态中，成员变量的特点

无论编译和运行，都参考左边（引用型变量所属的类）。

在多态中，静态成员函数的特点:

无论编译和运行，都参考左边